The mathematical reformulation of regular expression

Nice title!

Tu Luan

Advised by? Instructed by: Professor Steven Lindell

(Citation is still under implementation.) Yeah, citation should be added in the next draft:)

Formal language refers to a set of strings of symbols and rules that are specific to it. What's it? Formal language theory mainly studies the internal and syntactical structure pattern of formal language. In computer science particularly, formal languages are applied to define the grammar of programming languages and formalized natural languages in which concepts that are related to meanings or semantics, are represented by the words of those languages. To be more specific, the source code parsing of compilers of many modern languages involve formal language theory, where the formal language theory ensures the validity of an incoming piece of source code. Additionally, in theoretical computer science, decision problems in computational complexity are always defined in formal language. Besides the field of computer science, formal language theory is also widely used in linguistic and math. For example, Formal language theory is applied in mathematical logic where a formal system contains a formal language and a deductive apparatus.

Grammatically doesn't quite make sence, but I understand what you want to say mostly

Sentence too long to follow. Try separating into two sentences?

validity in which aspect?

The main subject of this paper is the regular expression, or regex, a subcategory of formal language. In formal language theory, regular expression is a sequence of characters with searching pattern.

Our main motivation for reformulation is that, during the last semester of study in regular expression, we found similarities and connections of regular expression and free algebra and

monadic second order logic. This inspired us to try to reformulate regular expression in a possible new way that is can be more accessible to students who have basic algebraic background but not much computer science background in formal language. Maybe explain why do we want that accessibility?

???

Regular Expression

The standard definition of regular expression goes as the following:

Given a finite alphabet Σ, there are groups of constants defined as regular expressions:

1.Empty set ∅, denoting a set without elements.

2.Empty string ε, denoting a string without characters.

3.a in Σ, denoting the set containing character a only.

Besides constants, there are several operations that can produce regular expression from the constants.

1.Concatenation: Given regular expression A and B, AB denotes the sets of strings that can be obtained by concatenating a string in A and a string in B: for example: A={“a”,”bc”}, B = {“r”,”yt”}. Then concatenation of A and B is AB = {“ar”,”ayt”,”bcr”,”bcyt”}

2.Alternation: Given regular expression A and B, A|B denotes the set of strings that can be obtained by set union of A and B. With the same example above, where A={“a”,”bc”}, B = {“r”,”yt”}, A|B ={“a”,”bc”,“r”,”yt”}

3.Kleene Star: Given regular expression A, A* denotes a set of strings that can be obtained by concatenating any finite number set of strings from the set described by A. A* contains Empty

string ε and it is closed under concatenation. Let A={“a”,”bc”}, then A* = {ε, “a”, “bc”, “abc”,”abca”.....}

Instead of just listing each concepts like a textbook. Try to restructure the background into paragraphs. Even better, try lead readers through.

Algebra and free algebra

I understand this from Abstract Algebra, but you might want to explain a bit. At least say it’s from Abstract Algebra so readers can search themselves

Free algebra is the noncommutative analogue of a polynomial ring since its elements may be described as "polynomials" with non-commuting variables.

define Ring?

If we have a ring R, we can get the polynomials over this ring in R[X1,X2,⋯,Xn]. These polynomials have n letters. Each term in a polynomial element could be thought of as a list of elements with their powers and some unit. Ultimately the point is that order does not matter.

What’s a unit?

However, for a free algebra, the idea is to do the same but now we take care with order.

I understand what you want to say. But try rewording it more formally like “On the other hand, free algebra take care of orders beyond xxx as in normal algebra.

Homomorphism

A Homomorphism is a structure-preserving map between two algebraic structures of the same type. To be more specific, the map preserves the operation of the structures. Nice description!

A map f: G → H between two groups is a homomorphism if for every am in G such that 0<m and m<=k,

f(φA(a1,a2,....ak)) = φB(f(a1),f(a2),.....,f(al))

This can be demonstrated by the following example:

f:A->B between two sets A and B of the same structure that multiplication (*) is one operation of that structure. Then we have f(x*y) = f(x)*f(y) for x, y in A.

Example:

φ: G → H where G = Z and H = Z_2 = Z/2Z is the standard group of order two,

by the rule: You should migrate to Latex but at least typeset your math equations

φ(x) = 0 if x is even;

φ(x) = 1 if x is odd;

To check if φ is a homomorphism over the operation of addition, we do the following steps:

Suppose we have x, y that are any values from Z.

1.if x and y are both even or both odd, then x + y is even. In this case φ(x + y) = 0. In the first case φ(x) + φ(y) = 0 + 0 = 0 and in the second case φ(x) + φ(y) = 1 + 1 = 0. 2.if one of x,y is even and the other is odd and x + y is odd. In this case φ(x + y) = 1 and φ(x) + φ(y) = 1 + 0 = 1.

In both case φ(x + y) = φ(x) + φ(y), so we get a homomorphism of function φ(x).

Monoid

Monoid is a algebraic structure with a single associative binary operation and identity element.

Suppose S is a set and @ is a binary operation SxS->S,then set S with binary operation @ is a monoid if it satisfies the two following axioms:

Associativity: For any x, y ,z in S, (x@y)@z = x@(y@z) holds.

Identity element: There exist an element e in S, such that for any element x in S, e@a=a@e=a hold.

For example:The natural numbers N can form a commutative monoid under addition and multiplication.

For a, b c in N,

Addition:

Associativity:(a+b)+c = a+(b+c) holds.

Identity element: 0+a = a+0 = a hold.

Multiplication:

Associativity:(a*b)*c = a*(b*c) holds.

Identity element: 1*a = a*1 = a hold.

Besides the example we can find in mathematics binary operations, a finite set of strings over a fixed size alphabet forms a monoid under string concatenation:

For a, b, c in a finite set of strings over a fixed size alphabet

String concatenation:

Associativity: (a+b)+c = a+(b+c)

Identity element: “”+a = a+”” = a hold.

Monoid Homomorphisms

A homomorphism between two monoids $(M, *)$ and $(N, \bullet)$ is a function $f : M \to N$ such that

- $f(x * y) = f(x) \bullet f(y)$ for all x, y in M
- $f(e_M) = e_N$, where *eM and eN are identity elements of M and N respectively.*

Now we have an example of monoid homomorphism:

Let $(M, *)$ above to be (string, string concatenation(+*)). Therefore the identity element here is the empty string since a+”” = “” +a=a for any a in string.

Let $(N, \bullet)$ above to be (Natural number, Natural number addition(+•)) Therefore the identity element here is 0 since a+0 = 0+a=a for any a in natural number.

Let x,y to be any strings.

If we consider the following function:

(x.length +* y.length)=(x +• y).length

Nice explanations using examples! But again typeset your math equations, otherwise it’s really hard to read and follow

From our knowledge of string manipulation, It is clear that if you take two strings, calculate the length of each of them, and add them together will result the same answer as you concatenate the two strings first, then calculate the length of the newly generated string.

Let f(z) = z.length, we have the following equations:

$f(x * y) = f(x) \bullet f(y)$ for all x, y in M

And it is clear that

f(eM) = eN, since the length of an empty string is zero.

Since both of the two axioms hold, we can conclude that f(z)=z.length is a String->N function that preserves the monoid structure. So it is a monoid homomorphism function.

Equivalence class

Definition

Let R be a binary relation in A.

(a) R is called reflexive in A if for all a $\in$ A, aRa.

(b) R is called symmetric in A if for all a, b $\in$ A, aRb implies bRa.

(c) R is called transitive in A if for all a, b, c $\in$ A, aRb and bRc imply aRc.

(d) R is called an equivalence on A if it is reflexive, symmetric, and transitive in A.

Let E be an equivalence on A and let a $\in$ A. The equivalence class of a modulo E is the set:

[a]E = {x$\in$ A|xEa}

Congruence Relation

Congruence relation is an equivalence relation on algebraic structure. If f:A->B is a homomorphism between two algebraic structures, then the relation R is defined by a1Ra2 iff f(a1) = f(a2) is a congruence relation.

We particularly use the congruence relation in the following ways:

Suppose we have **fL:string->bool**

$$fL(x) = \{1 \text{ if } x \in L \text{ or } 0 \text{ if } x! \in L\}$$

Now we define the congruence relation to be a1 R a2 iff fL(a1)=fL(a2). And therefore we have a equivalence class $[a]R = \{x \in string | fL(x)=fL(a)\}$, which means a and x are in the same equivalence class if and only if both of them either are strings in language L or not strings in language L.

Regular expression is the homomorphism of preimage of finite monoid algebra.

It seems like you only covers Introduction and Literature Review.

For this extended abstract, it's recommended to have Introduction, Motivation, Background and Related Work, Literature Review and eventually some Current Results

In general, feels like you might need to work harder on your structure. Also migrating to Latex will help you with the math typeset